

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a publisher's version.

For additional information about this publication click this link.

<http://hdl.handle.net/2066/36638>

Please be advised that this information was generated on 2018-07-07 and may be subject to change.

Computer Science at Kent

**Implementation and Application
of Functional Languages
19th International Symposium, IFL 2007**

Olaf Chitil (Ed.)

Freiburg, Germany, 27th-29th September 2007

Technical Report No. 12-07
September 2007

Published by the Computing Laboratory,
University of Kent, Canterbury, Kent, CT2 7NF, UK

Preface

The 19th International Symposium on Implementation and Application of Functional Languages (IFL 2007) is held at Freiburg, Germany, on the 27th to the 29th September 2007. Local organiser is the Programming Languages Group of the Department of Computer Science of the University of Freiburg.

IFL brings together researchers active in the area of functional programming, with an emphasis on the implementation and application of the same. IFL provides an annual open forum for researchers who wish to present and discuss new ideas and concepts, work in progress, preliminary results, etc. IFL has been held throughout Europe in the Netherlands, United Kingdom, Germany, Sweden, Spain, Ireland and Hungary. This year for the first time IFL is co-located with the International Conference on Functional Programming (ICFP). A record number of 44 papers have been submitted for these draft proceedings. By the time of printing 73 researchers had registered for attendance at the symposium.

Following tradition, two proceedings are to be published: the draft proceedings used at the symposium (this document), released as a technical report of the Computing Laboratory of the University of Kent, and the post-symposium proceedings based on revised papers. The draft proceedings are un-refereed and provide a useful reference to the delegates at the symposium. All participants who give talks at the symposium are invited to submit revised papers for review after the symposium, to normal conference standards. The post-symposium proceedings of selected revised papers will be published by Springer-Verlag in its Lecture Notes in Computer Science (LNCS) series.

Olaf Chitil
Programme Chair
University of Kent
September 2007

Local Organisers

Markus Degen
Peter Thiemann
Stefan Wehr

Supported by Deutsche Forschungsgemeinschaft (DFG)

Table of Contents

Termination and Complexity Bounds for SAFE programs	8
<i>Salvador Lucas, Ricardo Peña</i>	
Graph Parser Combinators	24
<i>Steffen Mazanek, Mark Minas</i>	
Encoding Iterators in Interaction Nets	40
<i>José Almeida, Ian Mackie, Jorge Sousa Pinto, Miguel Vilaça</i>	
Testing Erlang Refactorings with QuickCheck	55
<i>Huiqing Li, Simon Thompson</i>	
Call Graphs, Dominator Trees, and Lambda Lifting	71
<i>Marco T. Morazan, Ulrik Schultz</i>	
To Be or Not to Be ... Lazy	89
<i>Mercedes Hidalgo-Herrero, Yolanda Ortega-Mallén</i>	
The Structure of the Essential Haskell Compiler, or Coping with Compiler Complexity	107
<i>Atze Dijkstra, Jeroen Fokker, Doaitse Swierstra</i>	
XHaskell — Adding Regular Expression Types to Haskell	123
<i>Martin Sulzmann, Kenny Zhuo Ming Lu</i>	
Evaluating and Using a Grid-Enabled Parallel Haskell	139
<i>Phil Trinder, Abyd Al Zain, Kevin Hammond</i>	
Partial Parsing: Combining Choice with Commitment	140
<i>Malcolm Wallace</i>	
Functional Master-Worker Skeletons	152
<i>Jost Berthold, Mischa Dieterle, Rita Loogen, Steffen Priebe</i>	
Towards an Implementation of a Computer Algebra System in a Functional Programming Language	168
<i>Oleg Lobachev</i>	
Lazy Contract Checking for Immutable Data Structures	179
<i>Robert Bruce Findler, Shu-yu Guo, Anne Rogers</i>	
Haskell – Join – Rules	195
<i>Martin Sulzmann, Edmund Lam</i>	
Splitting and Merging Program Refactorings	211
<i>Christopher Brown, Simon Thompson</i>	
An Interpretation of Temporal Properties in Functional Programs	224
<i>Máté Tejfel, Tamás Kozsik, Zoltán Horváth</i>	

Approaches to Subtyping in Functional Languages	229
<i>Glenn Strong</i>	
On the Validation of Specifications used in Model-Based Testing	230
<i>Pieter Koopman, Peter Achten, Rinus Plasmeijer</i>	
Car Damage Subrogation Workflow — an iTask exercise	232
<i>Erik Zuurbier, Rinus Plasmeijer</i>	
Towards Open Type Functions for Haskell	233
<i>Tom Schrijvers, Martin Sulzmann, Simon Peyton Jones, Manuel Chakravarty</i>	
Transparent Ajax and Client-Site Evaluation of iTasks	252
<i>Rinus Plasmeijer, Jan Martin Jansen, Pieter Koopman, Peter Achten</i>	
Static Inference of Non-Monotonic Polynomial Sized Types	254
<i>Marko van Eekelen, Olha Shkaravska</i>	
Efficient, Modular Tries	258
<i>Frank Huch, Sebastian Fischer</i>	
FunSETL — Functional Reporting for ERP Systems	268
<i>Michael Nissen, Ken Friis Larsen</i>	
The Reduceron: Widening the von Neumann Bottleneck for Graph Reduction using an FPGA	290
<i>Matthew Naylor, Colin Runciman</i>	
Incremental Extension of a Domain Specific Language Interpreter	301
<i>Olivier Michel, Jean-Louis Giavitto</i>	
Generic Programming Combinators	318
<i>Sebastian Fischer, Frank Huch</i>	
Supero: Making Haskell Faster	334
<i>Neil Mitchell, Colin Runciman</i>	
Checking Dependent Types Efficiently	350
<i>Dirk Kleeblatt</i>	
HW-Hume in Isabelle	366
<i>Chunxu Liu, Greg Michaelson</i>	
Static Contract Checking for Haskell	382
<i>Dana Na Xu, Simon Peyton Jones, Koen Claessen</i>	
Debugging Lazy Functional Programs by Asking the Oracle	400
<i>Bernd Braßel, Holger Siegel</i>	
Uniqueness Typing Simplified	416
<i>Edsko de Vries, Rinus Plasmeijer, David Abrahamson</i>	
Tabular Expressions and Total Functional Programming	431
<i>Baltasar Trancón y Widemann, David L. Parnas</i>	

Positive Supercompilation for a Higher Order Call-By-Value Language	441
<i>Peter Jonsson, Johan Nordlander</i>	
The Simple Category of Modules	457
<i>Mikolaj Konarski</i>	
Polytopes & Polytypes: Generic Isosurfacing & Functional Programming	474
<i>Colin Runciman, David Duke, Rita Borgo, Malcolm Wallace</i>	
Meta⟨Fun⟩ — Towards a Functional-Style Interface for C++ Template Metaprograms	489
<i>Ádám Sipos, Zoltán Porkoláb, Norbert Pataki, Viktória Zsók</i>	
Speculative Inlining of Predefined Procedures in an R5RS Scheme to C Compiler ..	503
<i>Marc Feeley</i>	
Circuit Parallelism in Haskell Programs	519
<i>Andreas Koltes, John O'Donnell</i>	
On Implementing S-Net	531
<i>Clemens Grelck, Frank Penczek</i>	
From Contracts Towards Dependent Types: Proofs by Partial Evaluation	534
<i>Stephan Herhut, Sven-Bodo Scholz, Robert Bernecky, Clemens Grelck, Kai Trojahner</i>	
A Rational Simplifier for GHC	551
<i>Laszlo Nemeth</i>	
Amortizing the Cost of Commuting Conversions when Beta-Reducing Monadic Normal Forms and A-Normal Forms	552
<i>Olivier Danvy</i>	

Index

- Abrahamson, David, 416
Achten, Peter, 230, 252
Al Zain, Abyd, 139
Almeida, Jose, 40

Bernecky, Robert, 534
Berthold, Jost, 152
Borgo, Rita, 474
Brassel, Bernd, 400
Brown, Christopher, 211

Chakravarty, Manuel, 233
Claessen, Koen, 382

Danvy, Olivier, 552
de Vries, Edsko, 416
Dieterle, Mischa, 152
Dijkstra, Atze, 107
Duke, David, 474

Feeley, Marc, 503
Findler, Robert Bruce, 179
Fischer, Sebastian, 258, 318
Fokker, Jeroen, 107

Giavitto, Jean-Louis, 301
Grelck, Clemens, 531, 534
Guo, Shu-yu, 179

Hammond, Kevin, 139
Herhut, Stephan, 534
Hidalgo-Herrero, Mercedes, 89
Horvath, Zoltan, 224
Huch, Frank, 258, 318

Jansen, Jan Martin, 252
Jonsson, Peter, 441

Kleeblatt, Dirk, 350
Koltes, Andreas, 519
Konarski, Mikolaj, 457
Koopman, Pieter, 230, 252
Kozsik, Tamás, 224

Lam, Edmund, 195
Larsen, Ken Friis, 268
Li, Huiqing, 55
Liu, Chunxu, 366
Lobachev, Oleg, 168
Loogen, Rita, 152
Lu, Kenny Zhuo Ming, 123
Lucas, Salvador, 8

Mackie, Ian, 40
Mazamek, Steffen, 24
Michaelson, Greg, 366
Michel, Olivier, 301
Minas, Mark, 24
Mitchell, Neil, 334
Morazan, Marco T., 71

Naylor, Matthew, 290
Nemeth, Lazlo, 551
Nissen, Michael, 268
Nordlander, Johan, 441

O'Donnell, John, 519
Ortega-Mallen, Yolanda, 89

Parnas, David L., 431
Pataki, Norbert, 489
Pena, Ricardo, 8
Penczek, Frank, 531
Peyton Jones, Simon, 233, 382
Pinto, Jorge Sousa, 40
Plasmeijer, Rinus, 230, 232, 252, 416
Porkolab, Zoltan, 489
Priebe, Steffen, 152

Rogers, Anne, 179
Runciman, Colin, 290, 334, 474

Scholz, Sven-Bodo, 534
Schrijvers, Tom, 233
Schultz, Ulrik, 71
Shkaravska, Olha, 254

Siegel, Holger, 400
Sipos, Adam, 489
Strong, Glenn, 229
Sulzmann, Martin, 123, 195, 233
Swierstra, Doaitse, 107

Tejfel, Máté, 224
Thompson, Simon, 55, 211
Trancón y Widemann, Baltasar, 431
Trinder, Phil, 139
Trojahner, Kai, 534

van Eekelen, Marko, 254
Vilaca, Miguel, 40

Wallace, Malcolm, 140, 474

Xu, Dana Na, 382

Zsok, Viktoria, 489
Zuurbier, Erik, 232

Transparant Ajax and Client-Site Evaluation of iTasks

– Draft Version –

Rinus Plasmeijer, Jan Martin Jansen, Pieter Koopman, and Peter Achten

Radboud University Nijmegen, Netherlands

`rinus@cs.ru.nl`, `jm.jansen.04@nlda.nl`, `pieter@cs.ru.nl`, `p.achten@cs.ru.nl`

Abstract

The iTask system is a combinator library written in Clean which allows the specification of multi-user workflow systems for the web. The iTask system generates forms that have to be filled in and submitted by the user. Each user has a set of tasks that can be processed in any order. The submission of a form might terminate existing tasks or create new tasks for the user herself as well as for other users. As a consequence a single event can cause a very complex state change on the server and can effect the work of many other users.

The advantages of using a browser as interface to a workflow system created with the iTask library is that no software has to be installed at the client site and that the look and feel of the GUI is familiar to every user. A drawback of this architecture is that the response might become rather slow when there are many users and many tasks. For each and every event on the client a message is sent to the server over the world wide web. The server processes the event and generates a new web-page for the user containing all her new tasks. For the calculation of the set of new tasks, the state of all other tasks has to be examined. Due to the delay of the world wide web and the creation, transportation and rendering of the complete new page by the browser, the response of a workflow system can become relatively slow.

In this paper we present two solutions for dealing with this performance problem.

First we introduce a combinator ‘UseAjax’ that cause the workflow system to use Ajax technology for handling a (sub)task. This has as consequence that only a part of the web page is updated instead of the creation, sending and rendering of an entire new page. The advantage of this extension is not only a smoother reaction in the browser on changes being made. Also the efficiency for large workflow systems is commonly improved in this way because most of the time only a smaller, for this (sub)task relevant, part of the current state needs to be recalculated.

For the definition of the workflow system a single annotation ‘UseAjax’ is sufficient. The implementation of this feature in the `iTask` library requires a Java script that runs on the client as well as a call-back function that handles the event. For the implementation this requires the possibility to store `Clean` functions temporarily in a web page as well as the possibility to store them in a persistent store at the server site such as in a file or in a database.

The second extension is another annotation, ‘OnTheClient’, which allows client site evaluation of tasks.

Since no call at all has to be made to the server when such a task is evaluated, there is no web communication overhead anymore as is the case when Ajax technology is being used. For the implementation of this feature one needs to be able to execute the tasks specified in `Clean` in the browser at the client site. We realize this with an interpreter for `Clean` code running in the browser. Therefore `Clean` is compiled to `Sapl` and this code is loaded into the browser together with the compact and efficient `Sapl` interpreter. Of course, code interpreted by the `Sapl` interpreter running in the browser is not as efficient as the execution of compiled `Clean` code at the server. So, there is also an efficiency penalty when ‘OnTheClient’ is chosen instead of ‘UseAjax’.

By choosing one of the annotations, the programmer can define which evaluation method preferably should be used for a certain (set of) tasks.

Whenever evaluation ‘OnTheClient’ is not possible for some reason (e.g. when a database needs to be inspected on the server) the system can automatically decide to ‘UseAjax’ instead.